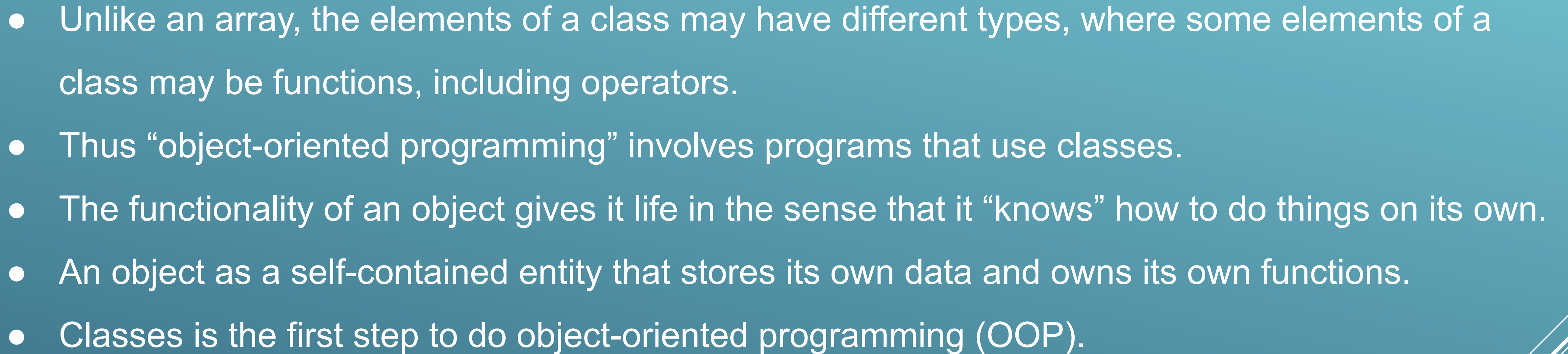
Programming 1 (C++)

Classes

INTRODUCTION TO CLASS



- Unlike an array, the elements of a class may have different types, where some elements of a class may be functions, including operators.
- Thus “object-oriented programming” involves programs that use classes.
- The functionality of an object gives it life in the sense that it “knows” how to do things on its own.
- An object as a self-contained entity that stores its own data and owns its own functions.
- Classes is the first step to do object-oriented programming (OOP).

CLASS DECLARATION

- Class Declarations :

class Ratio

name of the class

{ public:

void assign(int,int);

double convert();

void invert();

void print();

member function : designated as public

private:

int num,den;

member data : designated as private

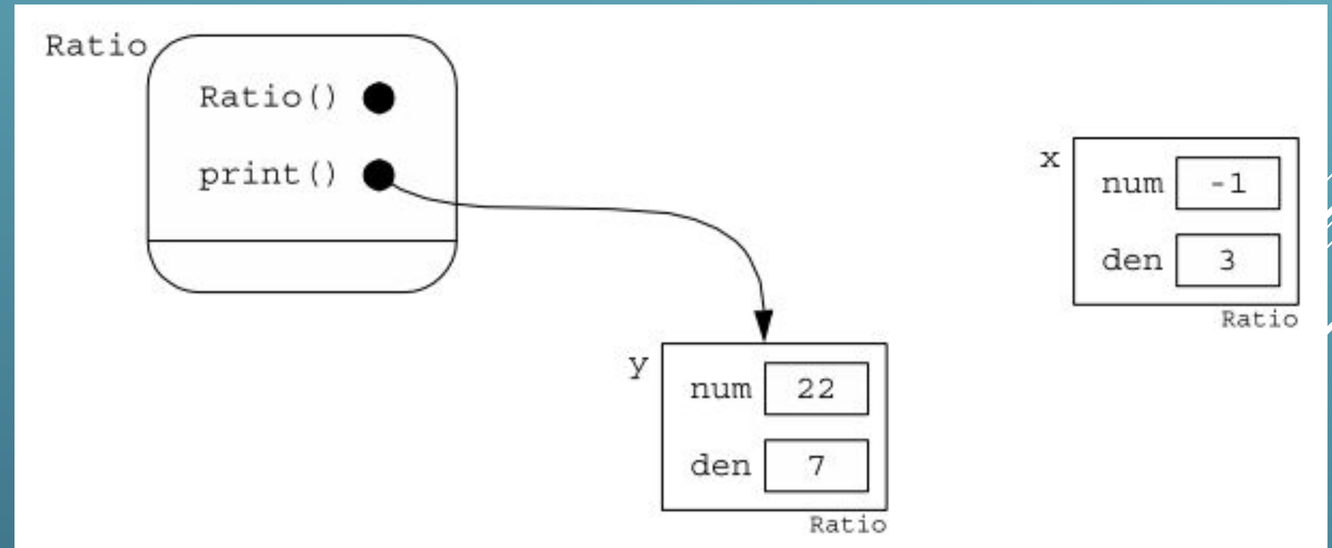
};

Preventing accessfrom outside the class is called “information hiding.” It allows the programmer to compartmentalize the software which makes it easier to understand, to debug, and to maintain.

CONSTRUCTORS

- A constructor is a member function that is invoked automatically when an object is declared.
- A constructor function must have the same name as the class itself, and it is declared without return type.

```
class Ratio
{ public:
Ratio(int n,int d) { num = n; den = d; }
void print() { cout << num << '/' << den; }
private:
int num,den;
};
int main()
{ Ratio x(-1,3), y(22,7);
cout << "x = ";
x.print();
cout << " andy=";
y.print();
}
```



Equivalent with 3 lines :

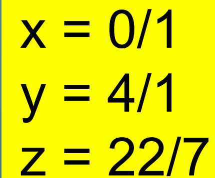
```
Ratio x,y;
x.assign(-1,3);
y.assign(22,7);
```

x = -1/3 and y = 22/7

CONSTRUCTORS

- Adding more constructors to the ratio class :

```
class Ratio
{ public:
Ratio(){num=0;den=1;}
Ratio(int n) { num = n; den = 1; }
Ratio(int n,int d) { num = n; den = d; }
void print() { cout << num << '/' << den; }
private:
int num,den;
};
int main()
{ Ratio x, y(4), z(22,7);
cout << "x = ";
x.print();
cout << "\ny = ";
y.print();
cout << "\nz = ";
z.print();
}
```



CONSTRUCTORS INITIALIZATION LISTS

C++ provides a special syntactical device for constructors that simplifies this code. The device is an *initialization list*.

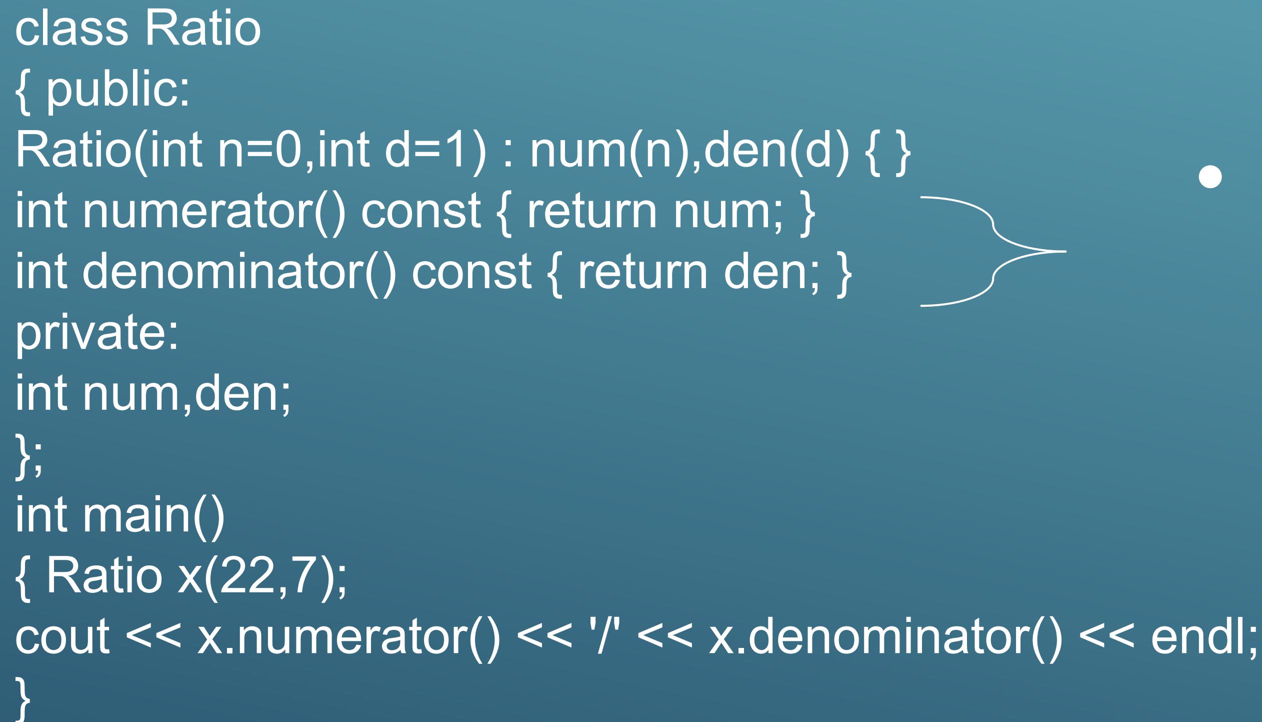
```
class Ratio
{ public:
Ratio(int n=0,int d=1) : num(n),den(d) { }
private:
int num,den;
};
int main()
{ Ratio x,y(4),z(22,7);
}
```

Initialization list.

x = 0/1
y = 4/1
z = 22/7

ACCESS FUNCTIONS

Although a class's member data are usually declared to be private to limit access to them, it is also common to include public member functions that provide read-only access to the data. Such functions are called *access functions*.



```
class Ratio
{ public:
Ratio(int n=0,int d=1) : num(n),den(d) { }
int numerator() const { return num; }
int denominator() const { return den; }
private:
int num,den;
};
int main()
{ Ratio x(22,7);
cout << x.numerator() << '/' << x.denominator() << endl;
}
```

- *return the values of the private member data.*
- *The use of the "const" keyword in the declarations of the two access function allows the functions to be applied to constant objects.*

PRIVATE MEMBER FUNCTIONS

- In some cases, it is useful to declare one or more member functions to be private.
- As such, these functions can only be used within the class itself; i.e., they are local *utility functions*.

```
class Ratio
{ public:
Ratio(int n=0,int d=1) : num(n),den(d) { reduce(); }
void print() const { cout << num << '/' << den << endl; }
private:
int num,den;
void reduce();
};
int gcd(int,int);
void Ratio::reduce()
{ // enforce invariant(den > 0):
if (num == 0 || den == 0)
{num=0;
den=1;
return;
}
if (den < 0)
{ den *= -1;
num *= -1;
}
// enforce invariant(gcd(num,den) == 1):
```

PRIVATE MEMBER FUNCTIONS

```
// enforce invariant(gcd(num,den) == 1):
if (den == 1) return; // it's already reduced
int sgn = (num<0?-1:1); // no negatives to gcd()
int g = gcd(sgn*num,den);
num /= g;
den /= g;
}
int gcd(int m,int n)
{ // returns the greatest common divisor of m and n:
if (m<n) swap(m,n);
while (n>0)
{ int r=m%n;
m=n;
n=r;
}
return m;
}
int main()
{ Ratio x(100,-360);
x.print();
}
```

THE COPY CONSTRUCTOR

- Every class has at least two constructors.
- These are identified by their unique declarations

```
X();
// default constructor
X(const X&); // copy constructor
```

- where X is the class identifier. For example, these two special constructors for a Widget class would be declared:

```
Widget();
// default constructor
Widget(const Widget&); // copy
constructor
```

- The first of these two special constructors is called the default constructor; it is called automatically whenever an object is declared in the simplest form, like this: Widget x;
- The second of these two special constructors is called the copy constructor;itiscalled automatically whenever an object is copied (i.e., duplicated), like this: Widget y(x);

CLASS DESTRUCTOR

- When an object is created, a constructor is called automatically to manage its birth.
- Similarly, when an object comes to the end of its life, another special member function is called automatically to manage its death.
- This function is called a destructor.

```
class Ratio
{ public:
Ratio() { cout << "OBJECT IS BORN.\n"; }
~Ratio() { cout << "OBJECT DIES.\n"; }
private:
int num,den;
};
int main()
{ { Ratio x;
cout << "Now x is alive.\n";
}
cout << "Now between blocks.\n";
{ Ratio y;
cout << "Now y is alive.\n";
}
}
```

CONSTANT OBJECTS

- It is good programming practice to make an object constant if it should not be changed.
- This is done with the const keyword:
  - const char BLANK='';
  - const int MAX_INT = 2147483647;
  - const double PI = 3.141592653589793;
  - void init(float a[], const int SIZE);
- Like variables and function parameters, objects may also be declared to be constant::
  - const Ratio PI(22,7);
- A function is declared constant by inserting the const keyword between its parameter list and its body:
  - void print() const { cout << num << '/' << den << endl; }

POINTER TO OBJECTS

- In many applications, it is advantageous to use pointers to objects

```
class X
{ public:
int data;
};
int main()
{X*p=new X;
(*p).data = 22;
// equivalent to: p->data = 22;
cout << "(*p).data = " <<
(*p).data <<"="<< p->data <<
endl;
p->data = 44;
cout << " p->data = " <<
(*p).data <<"="<< p->data <<
endl;
}
```

STATIC DATA MEMBERS

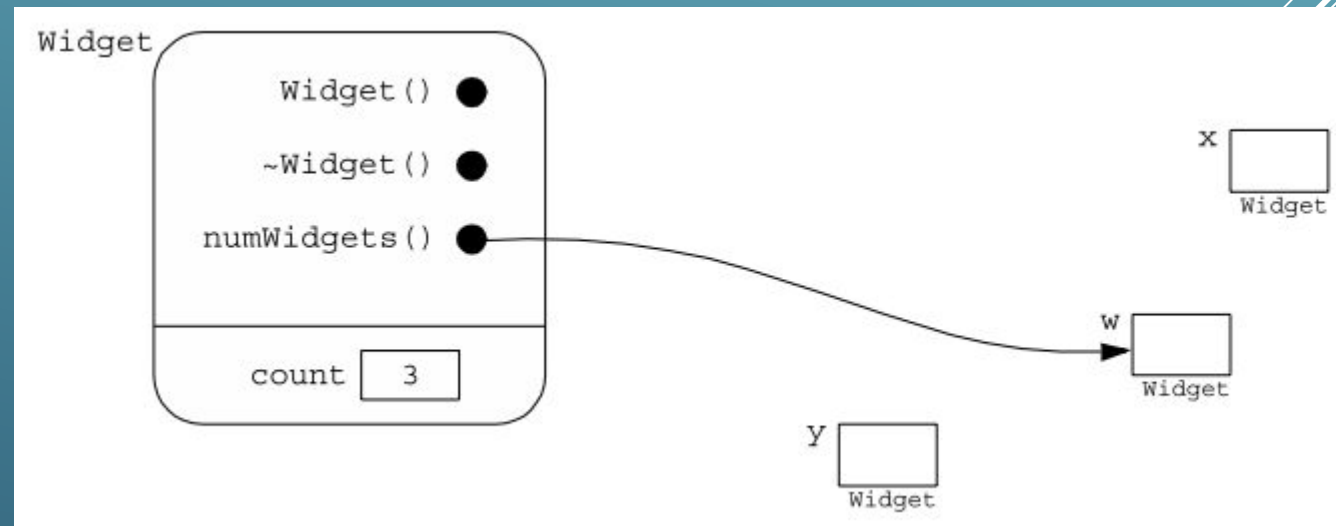
- Sometimes a single value for a data member applies to all members of the class.
- In this case, it would be inefficient to store the same value in every object of the class.
- That can be avoided by declaring the data member to be static.

```
class X
{ public:
static int n; // declaration of n as a static data member
};
int X::n = 0;
// definition of n
```

STATIC DATA MEMBERS

- The Widget class maintains a static data member count which keeps track of the number of Widget objects in existence globally.
- Each time a widget is created (by the constructor) the counter is iincremented, and each time a widget is destroyed (by the destructor) the counter is decremented.

```
class Widget
{ public:
Widget() { ++count; }
~Widget() { --count; }
static int count;
};
int Widget::count = 0;
int main()
{ Widget w,x;
cout << "Now there are " << w.count << " widgets.\n";
{ Widget w,x,y,z;
cout << "Now there are " << w.count << " widgets.\n";
}
cout << "Now there are " << w.count << " widgets.\n";
Widget y;
cout << "Now there are " << w.count << " widgets.\n";
}
```



TUGAS KELOMPOK

Online Bookstore Management System

Scenario Overview:
You are tasked with developing a simple online bookstore management system. The system should allow users to add books to their shopping cart, view the contents of their cart, and proceed with checkout. Additionally, the system should keep track of available book inventory and update it accordingly.

Assignment Components:

1. Class Design:
   - Design classes representing books, shopping cart, and bookstore.
   - Determine appropriate member variables and member functions for each class.
   - Constructor Implementation:

2. Implement constructors for each class to initialize object state.
   - Ensure proper initialization of member variables, including book details and shopping cart contents.

3. Access Functions:
   - Implement access functions (getters and setters) to manipulate class data.
   - Ensure appropriate encapsulation and data hiding.

4. Private Member Functions:
   - Implement private member functions to perform internal operations within classes.
   - Use private member functions for complex calculations or data manipulation.

5. Copy Constructors:
   - Implement a copy constructor for the shopping cart class to support copying cart contents.
   - Ensure proper deep copy of cart items.

THANK YOU